**TECHNICKÁ UNIVERZITA V LIBERCI**
Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B2612 – Elektrotechnika a informatika
Studijní obor: Elektronické informační a řídící systémy

# Transformace komplexních hradel v popisu obvodu na základní hradla

# Transformation of the complex gates in a circuit description into the simple gates

**Bakalářská práce**

Autor: **Pavel Jonáš**
Vedoucí práce: Ing. Jiří Jeníček, Ph.D.

**V Liberci 1. 2. 2009**

# Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé bakalářské práce a prohlašuji, že **souhlasím** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum

Podpis

## Poděkování

Na tomto místě bych rád poděkoval vedoucímu mé bakalářské práce Ing. J. Jeníčkovi, Ph.D. za rady, připomínky a čas, který mé práci věnoval. Dále bych chtěl poděkovat svým spolubydlícím na koleji za toleranci při mém nočním programování. A nakonec bych chtěl poděkovat své rodině za vytvoření potřebného zázemí pro mé vysokoškolské studium.

## Abstrakt

Cílem této práce je vývoj aplikace pro zpracování zdrojového kódu VHDL a jeho následná transformace na zdrojový kód BENCH. K pochopení problematiky je nutné mít alespoň základní znalosti týkající se VHDL popisu a znalost práce s textovým souborem. Práce seznamuje s potřebnými základy jak o VHDL, tak i FPGA, dále vysvětluje pojmy základní a komplexní logická hradla. Dále obsahuje popis práce samotné aplikace a možnosti jejího případného doplnění.



## Abstract

The aim of this work is to develop applications for the processing of VHDL source code and its subsequent transformation to the source code BENCH. To understand the problem it is necessary to have at least basic knowledge of VHDL description and knowledge of working with text files. This work presents the necessary fundamentals of the VHDL and the FPGA, further explains the concepts of basic and complex logic gates. It also includes a description of the work itself and the possibility of any additions.

# Obsah

# Seznam obrázků

# Seznam Tabulek

# Seznam použitých zkratek

| | |
|---|---|
| **VHDL** | Very High Speed Integrated Circuits Hardware Description Language |
| **FPGA** | Field Programmable Gate Array |
| **LUT** | Look Up Table |
| **MUX** | Multiplexor |
| **TTL** | Tranzistorově-Tranzistorová Logika |
| **CMOS** | Complementary Metal–Oxide–Semiconductor |
| **IEEE** | Institute of Electrical and Electronics Engineers |
| **RTL** | Register Fransfer Level |
| **ROC** | Reset On Configuration |
| **TOC** | sTate On Configuration |
| **EEPLD** | Electrically Erasable PLD |
| **AND** | Logický součin |
| **OR** | Logický součet |
| **NOT** | Logická negace |
| **NAND** | Negovaný logický součin |
| **NOR** | Negovaný logický součet |
| **XOR** | Exkluzivní logický součet |
| **NXOR** | Negovaný exkluzivní logický součet |

# Úvod

Cílem této práce bylo vytvořit program pro transformaci VHDL popisu, který dokáže přetransformovat komplexní hradla a jiné složitější podobvody na hradla základní. Uvažovaná komplexní hradla jsou z FPGA, tedy LUT a MUX ve všech svých variantách, jejich seznam je rozšiřitelný. Výsledný výpis musel být formátu BENCH, který je pak dále použitelný pro generování testů pro obvody na FPGA na Virtex4. Při přepisu se počet hradel několikanásobně zvětšil, proto bylo potřeba jejich počet snížit pomoci minimalizace.
Každá z těchto částí je popsána v následujících kapitolách.

- První kapitola popisuje, co je to VHDL.
- Kapitola druhá se věnuje popisu FPGA.
- Další kapitola obsahuje popis základních hradel.
- V následující kapitole je rozebrán popis hradel komplexních.
- V kapitole 5 se zabývám možnostmi formátu BENCH.
- Kapitola číslo 6 popisuje vznik samotného programu.

# 1 VHDL

## 1.1 Popis VHDL

Jazyk VHDL je jazyk navržený speciálně pro účely popisu a simulace velmi rozsáhlých číslicových obvodů a systémů (např. procesorů, čipsetů, programovatelných logických obvodů, aj.). Označení VHDL vzniklo jako akronym z názvu Very High Speed Integrated Circuits Hardware Description Language. Jazyk kombinuje rysy jazyka pro modelování a simulaci, jazyka návrhového, jazyka pro testování a jazyka pro popis topologie (netlist).

Jazyk VHDL je určen pro cílovou implementaci v reálném hardware.

Každý program vytvořený ve VHDL obsahuje tyto části:

- Komentáře
  - jsou označeny dvěma pomlčkami --
  - mohou se vyskytovat kdekoliv v programu, nemusí to být pouze na začátku řádku
  - končí vždy na konci řádku (VHDL nepodporuje víceřádkové komentáře)
- Library
  - zobrazuje použité knihovny
- Entity
  - popisuje rozhraní (vstupy a výstupy) objektu
  - definuje typ a směr přenosu dat
- Architecture
  - definuje vlastní chování a funkci entity
  - jedna entita může mít víc architektur
  - obsahuje deklaraci všech použitých signálů
  - Umožňuje tyto typy popisu: behaviorální, popis toku dat, RTL a **Strukturální**

## 1.2 Popis na úrovni struktury zapojení

Strukturální úroveň abstrakce (structure) je použita pro popis obvodu z hlediska jeho součástí. Struktura může být použita k vytvoření popisů velmi nízké úrovně (úroveň hradel) nebo naopak pro vysokou úroveň (blokový diagram). Na úrovni hradel mohou být spojovány základní logická hradla a klopné obvody do určité struktury a vytvářet tak obvod. To je často nazýváno netlist. Pro obvody vyšší úrovně jsou spojovanými komponentami větší funkční bloky. Strukturní popis je pak využit k rozčlenění do menších částí, se kterými se dá efektivně pracovat.

Při popisu struktury ve VHDL jsou k dispozici komponenty a konfigurace, které jsou velice užitečné pro řízení úrovně složitosti návrhu. Použití komponent může dramaticky zvýšit schopnost znovupoužití částí návrhů a umožňuje a podporuje hierarchický přístup k návrhu.

# 2 FPGA

## 2.1 Základní popis FPGA

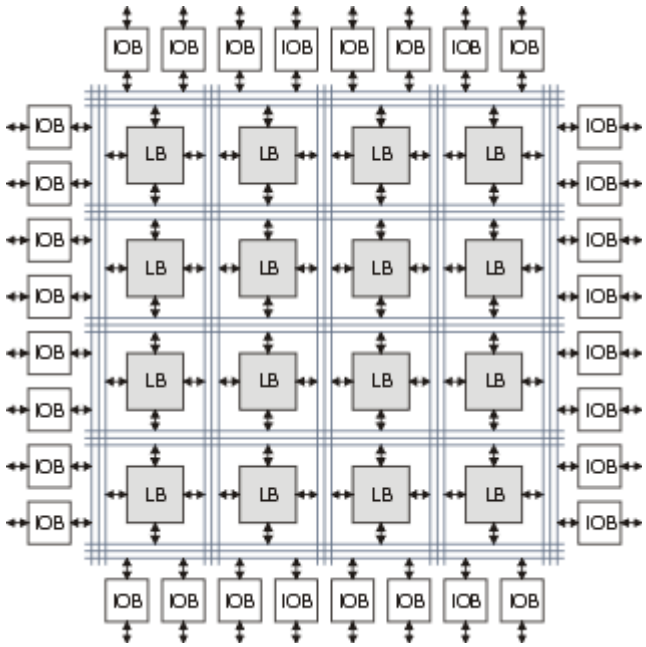
Pod zkratkou FPGA se skrývá označení Field Programmable Gate Array, což znamená programovatelné logické pole. Obvody postavené na architektuře FPGA patří do skupiny elektricky přeprogramovatelných PLD obvodů (EEPLD). Obvody architektury FPGA jsou založeny na malých generátorech logických funkcí s paměťmi, tzv. LUT tabulkách, dále pak klopných obvodech a mnoha horizontálních a vertikálních propojeních. Největší současné FPGA obvody obsahují několik stovek tisíc LUT tabulek a klopných obvodů. Kromě těchto bloků obsahují obvody FPGA specializované vstupně/výstupní bloky (IOB) s odpovídajícími zesilovači (výkonnými buffery) zajišťujícími dodržení logických úrovní používaných vně obvodu a potřebnou zatížitelnost, k nimž jsou připojeny vývody obvodu. U obvodů FPGA se obvykle konfigurace do obvodu přenáší vždy po připojení napájecího napětí z vnější připojené konfigurační paměti typu PROM či Flash nebo z jiného paměťového média. Konfigurační informace může být také obsažena například v paměti mikropočítače propojeného s obvodem FPGA, nebo se do obvodu FPGA dá zavádět z připojeného počítače typu PC. V současné době se však již setkáme i s jednočipovým řešením reprogramovatelných obvodů FPGA.

## 2.2 Skladba FPGA

### 2.2.1 Popis jednotlivých částí

Základ obvodů FPGA tvoří tyto stavební prvky:

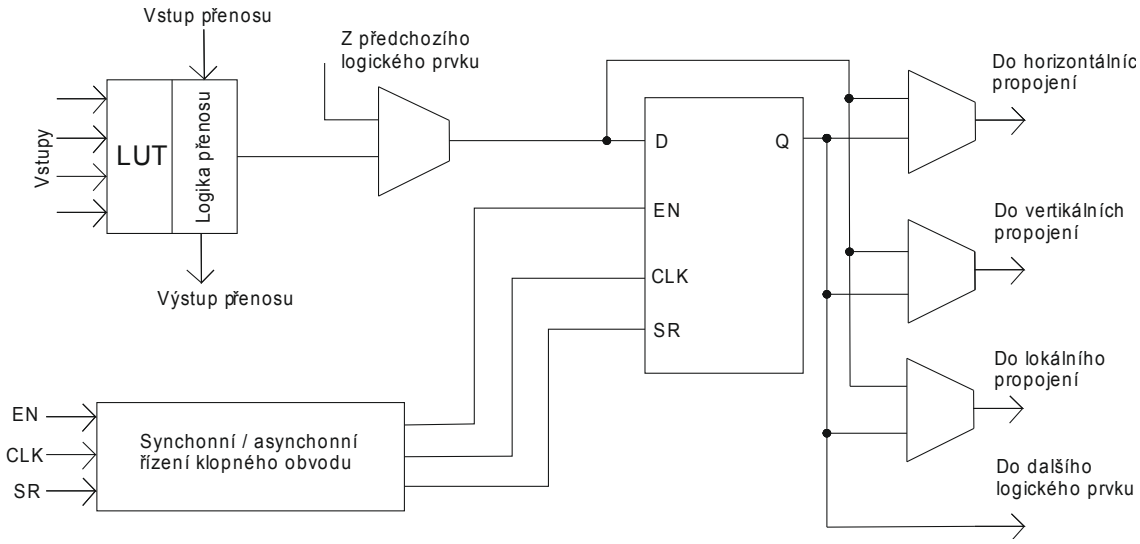
- Programovatelné **logické bloky** tvořené:
  - logickými prvky, které obsahují:
    - generátor logické funkce (vyhledávací tabulka LUT)
    - Klopný obvod
  - Lokálním propojovacím polem
- Programovatelné horizontální a vertikální **propojení**
- Programovatelné **vstupní/výstupní bloky** (I/O bloky)




*Obr. 1: Blokové schéma obvodu FPGA*

K těmto základním stavebním prvkům přibyly také tzv. specializované bloky, do nichž můžeme například zařadit paměti, násobičky, bloky pro úpravu hodinových signál, nebo dokonce celé procesory.

Výše jmenované stavební prvky mají u různých výrobců (např. Firma Xilinx, Altera, Lattice Semiconductor, Actel) různá označení, jejich funkce však bývá podobná. Logické bloky vždy musí obsahovat několik logických prvků a lokální propojovací pole, které tyto prvky dokáže vzájemně propojit a dále umožňuje jejich následné propojení s logickými prvky v okolních logických blocích.




*Obr. 2: Zjednodušené blokové schéma logického prvku*

### 2.2.2 Popis hradla LUT

Logický prvek obsahuje generátor logické funkce pomocí paměti (LUT tabulku). Dále obsahuje podpůrnou logiku pro vstup a výstup aritmetického přenosu pro případ, kdy je logický prvek použit pro realizaci čítače, sčítačky, apod. Logický prvek také obsahuje klopný obvod a blok pro řízení tohoto obvodu, ať už synchronního nebo asynchronního. V současné době používají některé nejnovější FPGA obvody LUT tabulky, které mají až šest vstupů a dva výstupy, místo dnes běžných čtyř vstupů a jednoho výstupu. U některých výrobců lze také navíc tyto LUT tabulky nastavovat do různých konfigurací, např. jako několik tabulek o třech až šesti vstupech.
Obvody FPGA dnes představují nejsložitější programovatelné obvody s nejvyšším stupněm integrace.

# 3 Základní logická hradla

Mezi základní logické hradla můžeme zařadit obvody AND, OR, NOT. Různými kombinacemi těchto základních hradel můžeme získat jakýkoliv obvod v booleovské logice. Abychom nemuseli používat pouze tato hradla, vznikla hradla jako jsou NAND, NOR, XOR, NXOR apod. Dnes jsou tato hradla zabudována v integrovaných obvodech, kde se většinou nachází několik hradel stejného typu. Pro výrobu hradel se používají technologie s jak bipolárními (TTL), tak i unipolárními tranzistory(CMOS).

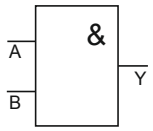
Všech kombinačních funkcí je $2^{2^N}$ kde N je počet vstupů. Počet úplně určených kombinačních logických funkcí pro dvě proměnné $2^{2^2}=16$ .

## 3.1 Popis hradel

### 3.1.1 AND

Hradlo AND obstarává logický součin všech vstupních signálů, této matematické operaci se říká konjunkce. To znamená, aby na výstupu byla logická 1 musí být všechny vstupy také na úrovni logické 1.

Rovnice pak má tedy tvar: $Y = A \times B$




*Obr. 3: Schématická značka hradla AND*

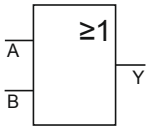
| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

*Tab 1: Pravdivostní tabulka hradla AND*

### 3.1.2 OR

Hradlo OR provádí matematickou operaci součet, té se také říká disjunkce. To znamená, že pokud chceme na výstupu hradla získat logickou 1, musíme přivést na logickou 1 alespoň na jeden ze vstupů hradla.

Rovnice logického součtu pak vypadá následovně: $Y = A + B$



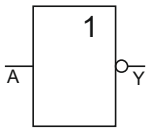

*Obr. 4: Schématická značka hradla OR*

| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

*Tab. 2: Pravdivostní tabulka hradla OR*

### 3.1.3 NOT

Hradlo NOT je úplně nejjednodušší logický obvod. Realizuje funkci logické negace. Což znamená, že na výstupu je opačná logická hodnota, než se nachází na vstupu hradla. Rovnice pak vypadá takto: $Y = \bar{A}$




*Obr. 5: Schématická značka hradla NOT*

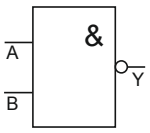
| A | Y |
|---|---|
| 0 | 1 |
| 1 | 0 |

*Tab. 3: Pravdivostní tabulka hradla NOT*

### 3.1.4 NAND

Toto hradlo vykonává funkci negovaného logického součinu. Je to nejběžněji používané hradlo. Propojením vstupů je schopno pracovat jako invertor (NOT). Pomocí hradla NAND lze realizovat všechny klopné obvody. *Tvoří úplný soubor funkcí*, tzv. Shefferovu funkci, tj. touto jedinou funkcí můžeme vyjádřit všechny kombinační logické funkce libovolného počtu proměnných.

Rovnice má tvar: $Y = \overline{A \times B}$




*Obr. 6: Schématická značka hradla NAND*

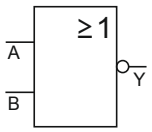
| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

*Tab. 4: Pravdivostní tabulka hradla NAND*

### 3.1.5 NOR

Hradlo obstarávající funkci takzvaného negovaného logického součtu. Aby se na výstup dostala logická jednička, je potřeba přivést logickou nulu na všechny vstupy hradla. NOR také *tvoří úplný soubor funkcí,* tzv. Piercovu funkci.

Rovnice má tvar: $Y = \overline{A + B}$




*Obr. 7: Schématická značka hradla NOR*

| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

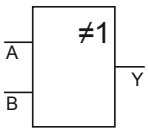
*Tab. 5: Pravdivostní tabulka hradla NOR*

### 3.1.6 XOR

Hradlo vykonává matematickou operaci známou jakou výhradní součet, který je též známý jako nonekvivalence (neshoda). Logická 1 je na výstupu pouze tehdy, když jsou na vstupech různé logické hodnoty.

Rovnice má tvar:

$$Y = A \oplus B$$




*Obr. 8: Schématická značka hradla XOR*

| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

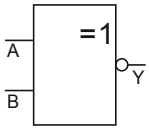
*Tab. 6: Pravdivostní tabulka hradla XOR*

### 3.1.7 NXOR

Hradlo NXOR je spolu s hradly NAND a NOT pouze negací svého předchůdce, v tomto případě XORu. Tedy plní funkci ekvivalence. Logická 1 je na výstupu pouze tehdy, když jsou na vstupech shodné logické hodnoty.

Rovnice má tvar:

$$Y = \overline{(A \oplus B)}$$




*Obr. 9: Schématická značka hradla NXOR*

| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

*Tab. 7: Pravdivostní tabulka hradla NXOR*

# 4 Komplexní hradla

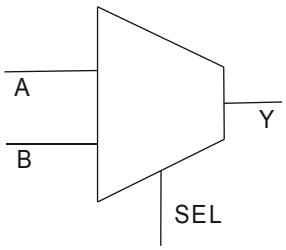
Komplexní hradla jsou složitější logické obvody, které mají svou vlastní specifickou funkci. Obvody tohoto typu jdou sestavit z hradel základních, ale jsou vetšinou integrovány již v pouzdrech jako hradla komplexní. Do těchto obvodů se dají zařadit například multiplexory a LUT tabulky.

## 4.1 Multiplexor

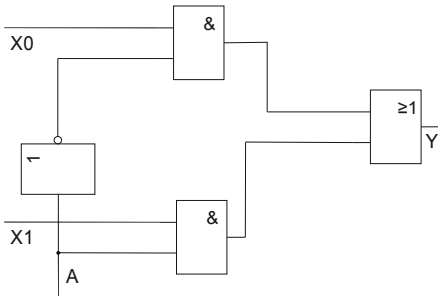
Multiplexor, označovaný jako MUX je kombinační logický obvod, který funguje podobně jako obyčejný vícepolohový přepínač. Obecně máme několik digitálních vstupů X, jeden výstup Y a tzv. adresový vstup A, který funguje jako „páčka“ pro výběr požadovaného vstupu. Hodnotou adresového vstupu určujeme, který vstup bude přiveden na výstup. Abychom multiplexor využili naplno, je potřeba mít dostatečný počet adresových bitů - jedině tak budeme moci přepínat mezi všemi možnými vstupy.

### 4.1.1 Příklad zapojení MUX2

Nejjednodušší možný multiplexor, což je 1-bitový multiplexor 2:1, se skládá z pouhých čtyř hradel - dvou hradel AND, jednoho hradla OR a z jednoho invertoru. Hradla AND mají za úkol spínat vstupní bity podle hodnoty adresového vstupu. Invertor se stará o to, aby se oba vstupy navzájem vylučovaly (na výstup musí být připojen vždy pouze jeden ze vstupů). A konečně hradlo OR spojuje obě hradla AND do jednoho výstupu.




*Obr. 10: Schématická značka multiplexoru*




*Obr. 11: Vnitřní schéma zapojení multiplexoru*

| A | X0 | X1 | Y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

*Tab. 8: Pravdivostní tabulka dvouvstupového multiplexoru*

| A | X0 | X1 | Y |
|---|---|---|---|
| 0 | 0 | X | 0 |
| 0 | 1 | X | 1 |
| 1 | X | 0 | 0 |
| 1 | X | 1 | 1 |

*Tab. 9: Zjednodušená pravdivostní tabulka dvouvstupového multiplexoru*

Jak je vidět, pravdivostní tabulka multiplexoru má oproti hradlům základním při stejném počtu vstupních signálů dvojnásobný počet řádků. U multiplexoru čtyřvstupového se **počet řádků** v tabulce **zvýší až na 64**. Proto se u těchto obvodů s úplnými pravdivostními tabulkami nepracuje.

## 4.2 LUT

Tabulky LUT jsou jsou ve skutečnosti vícevstupové, vícevýstupové univerzální logické bloky. V této aplikaci se vyskytují ve třech variantách: LUT2, LUT3, LUT4. Například LUT2 je hradlo se dvěma vstupy, jedním výstupem a inicializačním vstupem. Na inicializační vstup je přivedena hodnota v hexadecimálním tvaru udávající nám, které řádky LUT tabulky budou mít na výstupu logickou jedničku.
Bohužel nevíme dopředu, jaká hodnota na inicializační vstup přijde, proto by bylo nutné připravit pravdivostní tabulky pro všechny možnosti.

# 5 Bench formát

Bench formát, někdy také nazývaný ISCAS89, je formát zápisu obvodu, který nám hierarchicky definuje popis systému. Aby bylo možno obvod dále upravovat případně spojit s jiným systémem. Struktura vypadá následovně:

- jméno systému
- vstupy / výstupy
- popis funkce
- konkretizaci systému v jiném systému

V našem případě je to výstupní formát vytvořeného programu, ve kterém již byla komplexní hradla přetransformována pouze na hradla základní. Jsou zde již také vytvořeny signály pro
globální nulu a jedničku. Popis je realizován pouze pomocí hradel AND, OR, NOT a XOR dále pak BUF a DFF.

# 6 Program

Pro vytvoření transformátoru jsem si vybral programovací jazyk C# pro jeho rozšířenost a kompatibilitu s různými operačními systémy na trhu.
Protože jeho kód je jednoduchý na pochopení, nebude problém v případě potřeby rozšířit o další funkce.

## 6.1 Popis funkce

První funkcí programu je načtení dodaného zdrojového kódu. Ten je napsán v jazyce VHDL a dodáván jako textový soubor formátu txt. Protože se jedná o textový soubor, zvolil jsem pro jeho rozložení využít regulárních výrazů. Vytvořil jsem třídu *LineInfo*, jejíž instance obsahují a dále zpracovávají text získaný pomocí regulárních výrazů.

### 6.1.1 Parsování

Abych rozeznal jednotlivé bloky od sebe, vytvořil jsem si seznam konstant s názvem *Linetype*, který obsahuje konstantu pro každý typ VHDL struktury v textu. Při vytváření *LineInfo* musí být jedna z těchto konstant zadaná jako parametr konstruktoru. Provázání mezi konstantami *Linetype* a regulárními výrazy je uskutečňováno pomocí dvou polí. Obě pole musí mít stejný počet prvků. První pole obsahuje konstanty *Linetype*, druhé pak odpovídající regulární výrazy. *LineInfo* samo o sobě dokáže zpracovat většinu struktur VHDL souboru jako například library, comment, hradla. Pro složitější strukturu architecture bylo potřeba vytvořit specializovaného potomka *ArchitectureInfo*. Architecture se skládá ze dvou částí. První část obsahuje pouze deklaraci signálů, která nebyla pro výsledek programu důležitá. Pro druhou část jsem vytvořil také specializovaného potomka pojmenovaného *BlockInfo*.

V této části se nachází VHDL popis všech hradel použitých v obvodu. Následně probíhá parsování uvnitř bloku, které tuto část rozdělí na jednotlivá hradla. Postupným parsováním vznikne stromová struktura tvořená z objektu typu *LineInfo*, která přesně kopíruje strukturu zdrojového VHDL souboru.
V dalším kroku proběhne vytvoření textového řetězce, který obsahuje zápis pro všechna hradla ze zdrojového VHDL souboru.

```
Mcount_counter_reg_mand : X_AND2
  port map (
    I0 => counter_reg(31),
    I1 => clear_start_inv1_141,
    O => Mcount_counter_reg_mand_99
  );
```

*Obr. 12: Příklad definice hradla a jeho propojení v souboru VHDL*

```
Mcount_counter_reg_mand_99 = AND(counter_reg__31,clear_start_inv1_141)
```

*Obr. 13: Příklad definice hradla z obr. 12 ve výstupním formátu BENCH*

Vždy není nutné použít všechny vstupní parametry, které struktura VHDL obsahuje. Proto jsem stvořil třídu *LogicBlock*, ta u jednotlivých hradel vybírá vstupy, které používám při sestavování daného hradla do BENCH formátu. V této části programu byly modifikovány názvy jednotlivých signálů tak, aby se v jejich jméně nevyskytovala závorka, protože v BENCH není povolena. Ta byla nahrazena dvěma podtržítky.

### 6.1.2 Výsledek parsování

Takto přetransformovanou VHDL strukturu ukládám do textového souboru *out.txt* .Tento postup je výhodný ze dvou důvodů. Za prvé při vývoji programu nebylo třeba při každém spuštění programu znovu zpracovávat zdrojový kód VHDL, což je vzhledem k jeho složitosti časově velmi náročné. Za druhé může být soubor použit jinou aplikací nebo tento program může dále zpracovávat výstup z jiné aplikace.

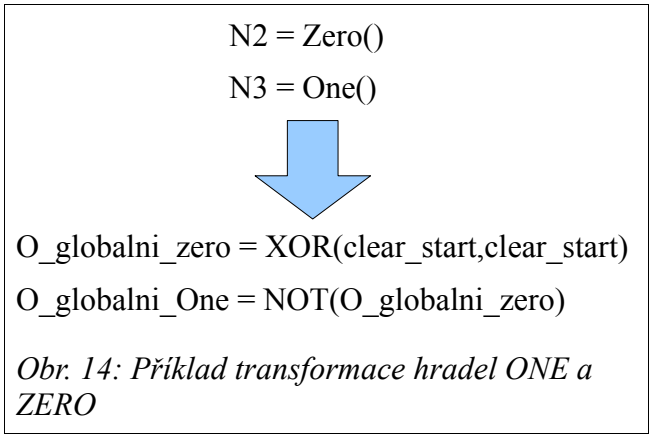
### 6.1.3 Transformace složitých VHDL hradel do požadované podoby

Výstupní soubor může obsahovat i hradla, které BENCH syntaxe nezná, proto musejí být přetransformovány do použitelné podoby. Soubor je znovu načten (*out.txt*) a jelikož se každé hradlo nachází pouze na jednom řádku, je následné parsování o mnoho jednodušší. Hradla jsou uložena v objektech typu *hradlo* a potomcích. Nejprve se v paměti vytvoří struktura složená z hradel a jejich propojení, ve chvíli kdy je struktura dokončena, začne program s jednotlivými úpravami. Třída *hradlo* mimo vstupních a výstupních parametrů obsahuje dvě důležité metody. Jedna je *PredzpracujVystup*, druhá se jmenuje *VytvorVystup*. Tyto metody slouží k modifikaci struktury hradel. *PredzpracujVystup* je určena k manipulaci s propojeními jednotlivých hradel (například k nahrazení nulových signálů globální nulou). *VytvorVystup* slouží k modifikaci struktury hradel (například nahrazení hradla MUX několika hradly AND a OR). Modifikace je potřebná pro tyto hradla:

- MUX ve všech svých podobách
- LUT ve všech svých podobách
- NXOR
- ROC, TOC, ONE a ZERO

### 6.1.4 Hradla ONE a ZERO

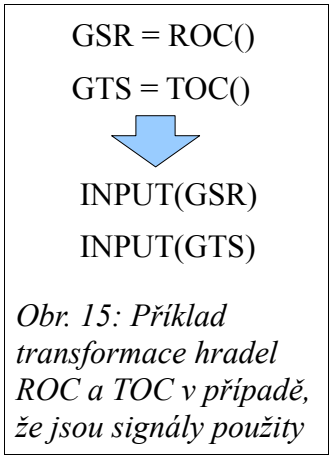
Hradla ONE a ZERO sloužící jako připojení napájecích signálů jsou přetransformovány na globální signály, které jsou vytvořeny pomocí libovolného signálu.




*Obr. 14: Příklad transformace hradel ONE a ZERO*

### 6.1.5 Hradla ROC a TOC

Hradla ROC(Reset On Configuration) a TOC (State On Configuration) jsou virtuální hradla vytvořená při kompilaci VHDL kódu. Jejich transformace může mít dvě podoby. Buď je výstupní signál z těchto hradel použit i u jiného hradla v obvodu, v tom případě je provedena transformace. Nebo je výstupní signál unikátní a nikde jinde se nevyskytuje, v tomto případě se tyto hradla do konečného BENCH formátu neuvádějí.

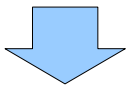



*Obr. 15: Příklad transformace hradel ROC a TOC v případě, že jsou signály použity*

### 6.1.6 Hradlo NXOR

Hradlo NXOR je nahrazeno kombinací hradel XOR a NOT, přičemž je potřeba vytvořit pomocný signál pro propojení mezi základními hradly, který musí být unikátní.

Výstup = NXOR(Vstup)



pomocny_Nxor_1 = XOR(Vstup)

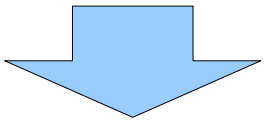
Výstup = NOT(pomocny_Nxor_1)

*Obr. 16: Příklad transformace hradla NXOR*

### 6.1.7 Hradlo MUX

U hradla MUX se dostáváme ke složitým transformacím. Při rozkládání na hradla základní musíme vytvořit hned několik unikátních pomocných signálů. Náhradní schéma je pak sestaveno z hradel AND, OR a NOT.

Vystup = MUX(Signal_A,Signal_B,SEL)



pomocny_mux_q1_0 = NOT(SEL)

pomocny_mux_q2_0 = AND(pomocny_mux_q1_0,Signal_A)

pomocny_mux_q3_0 = AND(Signal_B,SEL)

Vystup = OR(pomocny_mux_q2_0,pomocny_mux_q3_0)

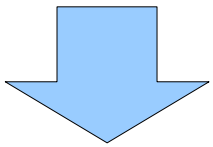
*Obr. 17: Příklad transformace hradla MUX se dvěma vstupními signály*

U multiplexorů s větším počtem vstupních signálů je výpis stejný, jen je zde větší adresní dekodér, tudíž je k transformaci potřeba většího počtu hradel NOT a AND, vše je pak sloučeno jedním hradlem OR.

### 6.1.8 Hradlo LUT

Transformace hradla LUT je nejsložitější operací v celé práci. Nejprve se vytvoří pravdivostní tabulka z inicializační hodnoty, která je poslední parametr ve vyparsovaném zdrojovém kódu. Jednotlivé řádky pravdivostní tabulky se získají pomocí bitových masek. Hodnoty z tabulky jsou nahrány do algoritmu řešícího minimalizaci logické funkce pomocí Karnaughových map. Pro každý typ Karnaughovy mapy (myšleno pro dvě, tři a čtyři proměnné) je napevno určené rozmístění stavových hodnot(A, B, C, D). Dále jsou určeny všechny typy minimalizačních smyček a pravidla jejich posunu po mapě. Program poté posunuje smyčky do všech možných pozic (od největší smyčky do nejmenší) a vytváří minimalizovanou logickou funkci. Tato funkce je pak složena pouze z hradel AND, OR a NOT a zapsána do výstupního souboru místo hradla LUT.

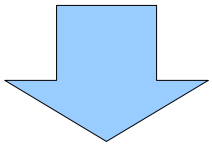
```
current_state_FSM_FFd3_In_F : X_LUT4 --
AKA:current_state_FSM_FFd3-In_F
  generic map(
   INIT => X"3CAA"   )
  port map (
   ADR0 =>  Vstup1,
   ADR1 =>  Vstup2,
   ADR2 => Vstup3,
   ADR3 => Vstup4,
   O => Vystup   );
```



```
Vystup = LUT4(Vstup1,Vstup2,Vstup3,Vstup4,X"3CAA")
```

*Obr. 18: Příklad transformace hradla LUT4 první část*

Vystup = LUT4(Vstup1,Vstup2,Vstup3,Vstup4,X"3CAA")



spoj_0 = AND(neg_0,neg_3)

spoj_1 = AND(neg_0,spoj_2)

spoj_2 = AND(neg_2)

spoj_3 = AND(neg_0,spoj_4)

spoj_4 = AND(neg_2)

spoj_5 = AND(neg_1,spoj_6)

spoj_6 = AND(neg_3)

spoj_7 = AND(neg_1,spoj_8)

spoj_8 = AND(neg_3)

Vystup = OR(spoj_0,spoj_9)

spoj_9 = OR(spoj_10)

spoj_10 = OR(spoj_11)

spoj_11 = OR(spoj_7)

neg_0 = NOT(Vstup1)

neg_1 = NOT(Vstup2)

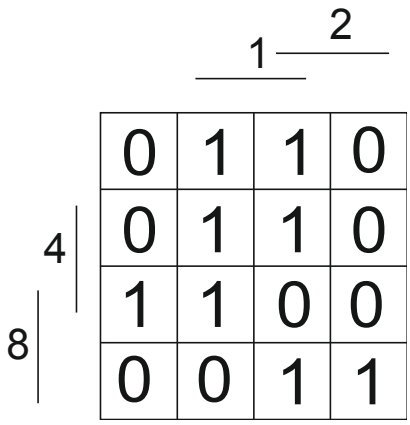
neg_2 = NOT(Vstup3)

neg_3 = NOT(Vstup4)

*Obr. 19: Příklad transformace hradla LUT4 druhá část*

| 3 | C | A | A |
|---|---|---|---|
| 0 0 1 1 | 1 1 0 0 | 1 0 1 0 | 1 0 1 0 |
| 15 14 13 12 | 11 10 9 8 | 7 6 5 4 | 3 2 1 0 |

*Tab. 10: Pravdivostní tabulka vstupní funkce hradla LUT*



*Obr. 20: Karnaughova mapa hradla LUT*

## 6.2 Minimalizace

Na minimalizaci logických funkcí se v praxi používají tři metody. Metoda Karnaughových map, která je názorná a jednoduchá pro funkce se čtyřmi a méně proměnými. Metoda Quine–McCluskey používá identický algoritmus pro jakýkoliv počet proměných, ale s jejich počtem exponenciálně narůstá náročnost výpočtu. Třetím minimalizačním systémem je Espresso algoritmus, vyvynutý firmou IBM. Algoritmus využívá heuristických pravidel, nezaručuje, že výstup bude minimální funkce, ale lze ho použít na logické funkce s mnohem vyšším počtem proměných než předchozí metody. Vzhledem k tomu, že nejsložitější komplexní hradlo, které se může nacházet ve zpracovávaném VHDL kódu, je čtyřvstupový LUT, tedy výčet stavů pro funkci se čtyřmi proměnými. Bylo tedy rozhodnuto, že pro minimalizaci použijeme metodu Karnaughových map.

### 6.2.1 Metoda Karnaughových map

Karnaughova mapa je upravený způsob zápisu pravdivostní tabulky, kde každá buňka mapy je přiřazení jednoho řádku pravdivostní tabulky pomocí stavových indexů, který nám umožňuje přímý zápis funkce v minimalizovaném tvaru. Mapa má tolik buňek, kolik má řádku pravdivostní tabulka. Stavové indexy v binární soustavě dvou vedle sebe ležících buňek se vždy liší pouze v hodnotě jedné vstupní proměnné (tzv. Grayův kód). Princip minimalizace spočívá ve vytváření smyček obsahujících $2^N$ logických jedniček.

Pravidla pro smyčky jsou:

- jedničky ve smyčce musí přímo sousedit
- smyčka musí mít tvar čtverce nebo obdélníku
- smyčka muže být i „přes hranu“ tabulky
- smyčky se mohou navzájem překrývat
- všechny jedničky v tabulce musí být obsaženy v nějakých smyčkách

Poté, co máme všechny jedničky obsaženy ve smyčkách, vypíšeme si součin vstupů, které jsou společné pro všechny jedničky ve smyčce. Pokud je některá ze společných vstupních veličin nulová, zapíšeme jí pomocí negace. Po vypsání všechny součiny sečteme a získáme tím minimalizovaný zápis původní funkce.

### 6.2.2 Implementace Karnaughových map

V programu se používá třída *Kmap2* pro reprezentaci Karnaughovy mapy se dvěma proměnými a její potomci *Kmap3* a *Kmap4* pro tři nebo čtyři proměnné. V každé třídě jsou definovány minimalizační skupiny a jejich posuny pro daný typ mapy. Samotná minimalizace probíhá tak, že se inicializuje tabulka logických hodnot ze vstupního vektoru hradla LUT (hexadecimální číslo), poté jsou nad všemi příslušnými skupinami prorevedeny všechny přípustné posuny. Pořadí, ve kterém jsou skupiny zpracovávány, je od největší po nejmenší. Pokud skupina po posunu obsahuje na všech pozicích logickou jedničku a alespoň některé pozice nejsou označené jako zpracované, je pozice skupiny uložena a políčka v mapě označena. Poté, co proběhlo ověření skupin se podle počtu skupin rozhodne, jak se LUT dále zpracuje. Pokud mapa neobsahovala žádnou skupinu jedniček, je jako výstup hradla LUT použita globální nula. Pokud pravdivostní tabulka obsahovala samé jedničky je výstup hradla LUT globální jednička. V ostatních případech je vygenerována logická funkce z hradel AND a OR viz. Oddíl 6.2.1

## 6.3 Implementace nového hradla do programu

Pokud chceme přidat do programu nové hradlo, musíme se řídit se řídit následujícími pokyny:

- Vytvořit regulární výraz
- Přidat typ hradla do seznamu konstant *Linetype*
- Doplnit do pole *exprs* a *types* na stejnou pozici regulární výraz a typ nového hradla (musí být umístěno před typy empty a unknown)
- Do třídy *LogicBlock* doplnit používané názvy vstupů nového hradla. Tato část deklarace stačí pro jednoduchá hradla, které se beze změn vstupů a výstupu vkládají do výstupního souboru (například AND, OR).
  Pro složitá hradla, kde je nutná manipulace se vstupy, výstupy nebo nahrazení souborem jiných hradel, musíme odvodit vlastní třídu s předkem *Hradlo*. V něm je pak potřeba zpracovat samotnou manipulaci.
- Ve třídě *hradlo* do konstrukční statické metody *CreateHradlo* doplnit vytváření třídy nového hradla.

## 6.4 Uživatelské rozhraní

Do prvního Editu se zadává vstupní soubor s VHDL popisem. Do druhého Editu je zadáván název souboru pro uložení rozparsování VHDL na jednotlivá hradla. Poté po kliknutí na tlačítko “ VHDL na pseudo BLOCK“ začne parsování a výsledek je uložen do souboru uvedeného v druhém Editu. Do třetího Editu zadáme název či cestu (defaultně ve složce Debug u programu), kam chceme výsledek uložit. Po kliknutí na tlačítko “ Pseudo BLOCK na BLOCK“ vytvoříme požadovaný textový soubor ve formátu BENCH.

# Závěr

Všechna hradla obsažené ve zdrojovém kódu VHDL byla přetransformována na jednoduchá hradla, zvolil jsem dvouvstupová základní hradla typu AND, OR, XOR či invertor NOT. Dále pak všechny typy bufferů byly přetransformovány na jeden společný tvar BUF. U přiloženého testovacího souboru se po minimalizaci počet základních hradel ve výstupním formátu BENCH zvýšil pětinásobně oproti vstupnímu formátu VHDL. Pokud by nebyla minimalizace použita, byl by tento nárůst hradel až třicetinásobný. Minimalizací hradla LUT ve všech jeho podobách jsme ušetřili přibližně 80% z celkového počtu hradel, které by byly potřeba na sestavení obvodu bez minimalizace. Zadání práce se tedy podařilo splnit.

# Seznam Použité literatury

[1] DRAYTON, Peter; ALBAHARI, Ben; NEWARD, Ted. *C# v kostce: Pohotová referenční příručka.* Praha : Grada , 2003. 788 s. ISBN 80-7300-198-5.

[2] ELLEN, Frank. *C#: Začínáme programovat.* Praha : Grada, 2002. 240 s. ISBN 80-247-0324-6.

[3] PINKER, Jiří; POUPA, Martin. *Číslicové systémy a jazyk VHDL.* Praha : BEN , 2006. 349 s. ISBN 80-7300-198-5

[4] PREDKO, Mike. *Digitální elektrotechnika bez předchozích znalostí : průvodce pro samouky.* Brno : Computer Press, 2008. 295 s. ISBN 978-80-251-2124-5.

[5] SATRAPA, Pavel. Katedra informačních technologií [online]. 2000 [cit. 2009-03-30]. *Regulární výrazy.* Dostupné z WWW: <http://www.kit.tul.cz/~satrapa/docs/regvyr/all.html>.

[6] SELLS , Chris. *C# a WinForms : programování formulářů windows.* Brno : Zoner Press, 2005. 645 s. ISBN 80-86815-25-0.